

A GENERIC CLUSTER AWARE LOCK BROKER WITH USER DEFINED LOCKING MODES

Technical Field

2 The present invention relates to the field of controlling access to devices on
3 clustered computer systems. More specifically, the present invention relates to the
4 brokering of locks utilized on a cluster to control access and/or usage of particular devices
5 and/or elements of a clustered network of computer systems.

Background

7 Commonly, pluralities of computers, databases, printers and other computing or
8 computing related devices are often established in clusters or as members or elements of a
9 network, hereafter, collectively “Clusters”. Clusters are often defined as a parallel or
10 distributed system that consists of a collection of interconnected whole computers, that is
11 utilized as a single, unified computing resource. As such, it is commonly appreciated that
12 Clusters commonly consist of computing devices (i.e., nodes) and other peripheral
13 devices to which access thereto may be controlled by particular nodes. Clusters enable
14 information system managers to share a computing load over several systems, thereby
15 enhancing the capabilities of the system as a whole, minimizing the adverse effects of
16 failures of certain nodes or devices on a Cluster, and allowing for the expansion of
17 capacity and capabilities of a given Cluster by adding additional or different nodes and/or
18 devices to the Cluster.

19 As Clusters have increased in size and complexity, interest has arisen in processes
20 for managing the utilization of resources associated with a Cluster (for example, printers,
21 processors, data storage devices, data files, displays and other virtual and real resources).
22 Throughout this description a device, whether real or virtual, accessible or utilized by a
23 Cluster shall be referred to hereinafter as a "Resource". Similarly, a computing device
24 (for example, a processor, server, mainframe computer or similar device) utilized in the
25 utilization of itself or other Resources, which may be directly or indirectly connected to
26 or accessed by such a computing device, shall be referred to hereinafter as a "Node".
27 Those skilled in the art appreciate that Resources and Nodes can include practically any
28 electronic device and be configured in any configuration desired.

29 Additionally, many large Clusters, in addition to managing access to Resources,
30 have to ensure that sufficient Resources are highly available (i.e., the Resources are
31 guaranteed to have a minimum operating reliability often measured in the 99.999% and

1 better ranges). As such, efficient and effective processes for managing the utilization of
2 Resources are highly desirable.

3 Currently, one method for managing Resources on a Cluster utilizes databases that
4 are maintained by every Node on the Cluster. These databases keep track of which
5 Resources are being utilized at any particular time by any client throughout the Cluster.
6 As is commonly appreciated, a client may be another Resource on a Cluster, a device
7 external to the Cluster (for example, a web browser on a user's personal computer (i.e., a
8 Node) connected to an Internet based Cluster), or anything else that requires access to a
9 Resource associated with a Cluster. Those persons, Resources, Nodes, Clusters and/or
10 any other entity that desires to utilize a Resource on a Cluster are hereinafter collectively
11 referred to as a "Client".

12 Since Clusters can often be quite large with numerous Resources being utilized by
13 widely dispersed Clients, keeping such databases up to date is often an inefficient and
14 inaccurate process. For example, in order to determine whether a Resource is currently
15 being utilized and then to reserve a Resource, the multi-node database approach often
16 requires a Node to keep track of every lock maintained on the Cluster. When a Client
17 desires to lock-up a Resource, the Client's Node then has to determine whether any other
18 Node has a lock on the Resource via its look-up tables. Further, when a lock is released,
19 the Node then has to notify every other Node on the Cluster of the releasing of the lock.
20 At which instance, every Node on the Cluster must update their respective lock tables.
21 As such, this approach can be very time intensive for all the Nodes on a Cluster. Thus, a
22 better process is needed.

23 Another process commonly utilized to reserve and/or "lock-up" a Resource is to
24 utilize a proxy maintained by a proxy server identified for the Resource. This approach
25 eliminates the need for each Node on the Cluster to maintain up-to-date databases or
26 listings identifying the utilization of every Resource on the Cluster. Instead, a single
27 Node provides the aforementioned proxy services. Usually, but not always, this Node is
28 the Node by which the specific Resources are accessed and/or controlled. However, this
29 approach is often undesirable because control of the Resource is often maintained by a
30 single Node. If the Node controlling a Resource fails, for whatever reason, often the jobs
31 to be processed by the Resource (which commonly are located on a proxy queue in a local
32 memory device associated with the Node) are lost and are not recoverable. As such an
33 efficient and reliable process for determining whether a Resource on a Cluster is available
34 and "locking-up" the Resource, when available, is needed.

Summary

The technical field provides a process for implementing a generic Cluster wide lock broker that allows various modes of locking up Resources on the various Nodes of a Cluster. The process utilizes a daemon, a CMLOCK daemon, to control the requesting, keeping and releasing of locks on Resources. The CMLOCK daemon is implemented on each "up" Node of the Cluster and communicates with other CMLOCK daemons on the other Nodes of the Cluster via a Cluster-wide broadcast communication mechanism. For "down" Nodes the CMLOCK daemon is suitably halted, thereby preventing "down" Nodes from locking up Cluster Resources.

The CMLOCK daemon controls the “locking-up” of Resources by utilizing a request that consists of three components, a lock name, an intent mode and a deny mode. The lock name identifies the Resource for which the request is submitted. The intent mode identifies those modes in which the Client desires to “lock-up” the Resource. The deny mode identifies those modes in which the Client desires to deny requests from peers for the same lock. The deny mode is operative only upon any Client obtaining the lock on the Resource. It is to be appreciated that multiple Clients may maintain a lock simultaneously on a Resource if their intent mode and deny mode support such concurrency. The CMLOCK daemon on each Node tracks the status of locks on Resources by using a Lock Table specific to each Node. By examining responses from other Nodes (identifying whether a particular lock on a Resource has already been obtained by a Client for that Node), a particular CMLOCK daemon can determine whether a Resource is available and reserve (“lock-up”) the Resource without needing to maintain records of every lock initiated by every other Client and/or Node and without having to inform every Node whenever a lock is established and/or removed. Further, the process does not rely upon proxies or other single point failure systems for determining when a Client may access a Resource.

In a second embodiment, the process also provides the capability of viewing locks currently held by other Clients on other Nodes. This embodiment utilizes the Cluster-wide broadcast communications capabilities of the CMLOCK daemon to request from other "up" Nodes those locks the CMLOCK daemons have established on each of such Nodes. As such, this process also provides a quick and efficient process for determining and monitoring Resource utilization on a Cluster.

Additionally, the process of the present invention provides the capability for a Client to establish pending lock requests, which enable the Client, via their host Node and

- 1 associated CMLOCK daemon, to re-request a lock on a Resource when a previously
- 2 existing lock (which prevented the requesting Client from gaining the desired access to
- 3 the Resource) is released. The Node establishes the denied lock in a pending state.
- 4 Utilizing the Cluster-wide broadcast communications capabilities, a CMLOCK daemon
- 5 “locking-up” a Resource may notify all other CMLOCK daemons on “up” Nodes for the
- 6 Cluster whenever a lock held by the daemon is released. Unlike a proxy or queue system,
- 7 wherein the Node associated with a Resource establishes a queue of requests, the present
- 8 invention does not utilize a queue and instead merely notifies all Nodes of a specific lock
- 9 being released. The Nodes interested in the Resource may then attempt to lock up the
- 10 Resource with the first Node to request obtaining the lock on the Resource.

11 As such, the technical field provides a process for determining the availability and
12 locking up of Resources on a Cluster without requiring all Nodes on a Cluster to monitor
13 the status of the various Resources on the Cluster and without requiring the Cluster to
14 utilize proxies, queues or similar techniques to assign a currently locked Resource to a
15 subsequent Client. The capabilities, process flows, and other specifics of the present
16 invention and its various embodiments are further described herein in the following
17 specification, claims and drawing figures.

Description of the Drawings

Figure 1 is a schematic representation of one Cluster of computing devices for which a lock brokering processes of the present invention may be utilized:

Figure 2 is a flow diagram of one embodiment of the lock brokering process of Fig. 1.

Figure 3 is a flow diagram of one embodiment of a process provided by the present invention for utilizing lock names, intent modes and deny modes; and

Figure 4 is a flow diagram of an alternative embodiment of the lock brokering process.

Detailed Description

The technical field a process for managing the utilization of Resources on a Cluster. The process utilizes a daemon, a CMLOCK daemon, to perform such operations. As is commonly known, a daemon is a process that runs in the background and performs a specified operation at predefined times or in response to certain events. Often a daemon is interpreted as being a process that is implemented in a UNIX® environment/operating system. However, in the this application, the term daemon is to be construed as being a process that runs in the background of any operating system and is not to be construed as

1 being limited to the UNIX environment. As such, other environments, including, but not
2 limited to, MICROSOFT® WINDOWS®, APPLE® MACINTOSH®, and other
3 environments, may be utilized. Typical daemon processes include print spoolers, e-mail
4 handlers, and other programs that perform administrative tasks for the operating system.
5 In short, any environment or operating system, which is capable of implementing
6 background processes and supports communications between the various Nodes of the
7 Cluster(s), may be utilized in conjunction with the present invention.

8 Figure 1 illustrates one embodiment of the process being implemented on a three
9 Node Cluster. As shown, a Cluster 100 includes Node A 102, which is connected to
10 Resources Printer A 104 and D-base A 106. Further, the Cluster 100 includes Node B
11 108 which is connected to a modem 110 and a fax machine 112. Additionally, Node C
12 114 is included in the Cluster 100 and is attached to PC#1 116 and PC#2 118. Each Node
13 is connected to each other Node in the Cluster 100 by suitable communications links 120,
14 122, and 124. It is to be appreciated that Nodes in a Cluster or other network
15 configuration may be connected utilizing various connectivity schemes including, but not
16 limited to, utilizing hubs, routers, star configurations, serial connections, parallel
17 connections, token ring connections, Ethernet configurations, Internet connections and
18 numerous others. This application is not directed to how Nodes in a Cluster are
19 connected and merely specifies that some connection between Nodes on a Cluster or
20 multiple Clusters is provided.

21 Further, it is to be appreciated that any Node on a Cluster may be suitably
22 connected and/or utilized to control any type or number of Resources. Those Resources
23 shown in Figure 1 are for purposes of illustration and explanation only and are not to be
24 construed as limiting any of the embodiments of the present invention to such a
25 configuration. Similarly, it is to be appreciated that physical Resources, such as a D-base
26 106 and/or the PCs 116/118 may be broken into various logical as well as physical
27 devices. Each such logical and/or physical device may be considered a Resource in and
28 of itself for whose control various Clients may request, monitor and/or obtain.

29 The CMLOCK daemon is appropriately loaded onto each of the Nodes. The
30 CMLOCK daemon creates a table, a Lock Broker Table (126, 128 and 130), as shown
31 next to each Node, which is utilized to determine which locks have been requested by a
32 Client utilizing a particular Node (for example, one using the PC#1 116 Resource
33 connected to Node C 114). Locks are registered by the CMLOCK daemon on a single
34 Node's Lock Broker Table in either a pending state or an active locked state. As shown

1 in Figure 1, the Node A Lock Broker Table 126 is holding two locks, a pending lock
2 request for Printer A 104 (the pending status is designated in Figure 1 by the "T" under
3 the pending column, whereas an "F" indicates an active lock) and a pending lock request
4 for PC#1 116. Further, the Lock Broker Table 126 identifies the client, by a Client ID,
5 requesting the lock for the particular Node. The CMLOCK daemon utilizes the Client ID
6 when it needs to notify a client that the lock operation was or was not successful.
7 Similarly, the Node B Lock Broker Table 128 is holding an active lock on PC#2 118, and
8 the Node C Lock Broker Table 130 is holding active locks on PC#1 116 and the fax 112.

9 As discussed previously, the present invention allows a Client to create lock
10 requests for the various resources available on a Cluster. Lock requests can also be
11 requested by Nodes and/or Resources without requiring the interaction or direction of a
12 Client. Regardless of whether a lock request is generated by a Client and/or a
13 Cluster/Node/Resource, each lock request includes three elements: a lock name, an intent
14 mode and a deny mode.

15 The lock name is unique to each Resource requested and each lock requested.
16 The lock name is utilized to identify Resources desired to be accessed and/or utilized by a
17 Client from a Node. The lock request is generated by a Client connected to a particular
18 Node (for example, Node C) and is only stored (in either an active or a pending state) in
19 the Lock Broker Table associated with the particular Node. So that all the Nodes on a
20 Cluster may be able to determine whether a particular Resource is locked and/or being
21 utilized in a particular manner (as designated by the intent and deny modes), the lock
22 request preferably includes in the lock name an identification of the specific Resource.

23 Further, it is commonly appreciated that Resources connected to a Node are often
24 identified by an Interrupt Request number. Similarly, Resources connected to a Cluster
25 or other network are often identified by a unique address, for example, an Internet
26 Protocol address. The present invention may be utilized in conjunction with any
27 addressing or identification scheme desired, provided that the Resources are specifically
28 identified as necessary, so that multiple contentions for the same Resources, at the same
29 time are prevented. Further, the lock name may be of any length and/or format desired,
30 but should be of such length so as to specifically identify the Resource being locked.

31 In addition to providing a lock name, a lock request also includes an intent mode.
32 The intent mode provides an indication of the manner in which the Client, or the
33 requestor of the lock, desires to access the Resource. The intent mode is preferably a 32
34 bit integer that indicates on a bit-wise basis how a Resource is desired to be utilized by a

1 client. In the tables 126, 128, 130, shown in Figure 1, four bits are depicted for
2 illustrative purposes, however, it is to be appreciated that any number of bits may be
3 utilized. In short, the intent mode provides an indication to all the other Nodes on the
4 Cluster that the Client desires to utilize a particular Resource in a particular manner. For
5 example, a Client may desire to utilize a data file (for example, a word processing
6 document) such that it is read only (i.e., other Clients can access the file because the
7 Client has requested a read only lock). Similarly, another Client may request to lock up
8 the same data file such that only they can write to the file. Basically, the intent mode
9 specifies how the Client desires to utilize a particular Resource.

10 The request also includes a deny mode. As provided for the intent mode, the deny
11 mode utilizes an integer number to represent, on a bit-wise basis, how a lock holder of a
12 Resource will allow other Clients to utilize the Resource. As with the intent mode, 32
13 bits are preferably utilized. However, the deny mode, for example, may exclude all
14 Clients from accessing the Resource, may allow certain Clients to utilize the Resource
15 while denying others, or may utilize any other schema desired. For example, a deny
16 mode might be configured to allow other Clients to access a data file (currently locked by
17 the first Client/lock holder) on a read-only basis while denying access to the data file to
18 any Client who desires to write to the data file. Such a lock might be called a read/write
19 lock.

20 The process, by which the present invention determines whether a subsequent
21 Client may gain access to a currently “locked” Resource, utilizes two bit-wise exclusive
22 OR comparisons. In the first comparison, the intent mode specified in a lock request is
23 compared to the deny mode for an active lock currently held by a Client. In the second
24 comparison, the intent mode specified by the active lock is compared to the deny mode
25 specified in the lock request. When the results of all the bits compared are “YES”, the
26 request is approved. Similarly, when the result of any of the bits compared is “NO”, the
27 request is denied. For the first embodiment of the present invention, a denied request is
28 cleared from the Lock Broker Table for the Node upon rejection or termination of the
29 request.

30 More specifically, the process by which a Client requests a lock, and the
31 CMLOCK daemon determines whether such lock request may be granted is illustrated in
32 Figure 2. As shown, this process begins with a first Node receiving a lock request from a
33 Client (Block 202). For purposes of explanation, the Node receiving the request is
34 referred to herein as the “First Node”. Those other Nodes on the Cluster(s) are referred to

1 herein as the “Peer Nodes”. The process flows illustrated in Figures 2-4 are depicted with
2 reference to the First Node and one Peer Node. It is to be appreciated, however, that the
3 process steps may be repeated and/or expanded to encompass as many Peer Nodes as are
4 necessary to determine whether a lock can be placed on a Resource associated with a
5 Cluster.

6 Further, it is to be appreciated that the First Node may receive the lock request via
7 any known techniques by which Resources on a Cluster are locked. Those skilled in the
8 art appreciate the various techniques, systems, and methodologies which may be utilized
9 to request a lock including, but not limited to, a request from a Client (for example, a
10 request to utilize a printer) or a request by a web browser for a connection with a specific
11 network server.

12 Upon receiving the lock request, the First Node, via the CMLOCK daemon,
13 suitably stores the lock request as a pending request in its associated Lock Broker Table
14 (Block 204). As previously discussed hereinabove, two examples of pending lock
15 requests are shown in Figure 1 by the italicized entry for *Printer A* or the entry for *PC#1*.
16 The pending state exists until every “up” Node responds to the request with either an
17 approval or a denial, as discussed in greater detail hereinbelow.

18 Next, the First Node communicates the lock request to all the Peer Nodes via the
19 Cluster-wide communications systems (Block 206). When communicating the request,
20 the First Node sends the lock name, the intent mode and the deny mode. For example, the
21 pending request in Figure 1 for Printer-A would be sent to the Peer Nodes (i.e., Node B
22 108 and Node C 114) as a request in the following format:

<u>Name</u>	<u>Intent Mode</u>	<u>Deny Mode</u>
Printer A	1011	1111

23 The name indicates the Resource requested, i.e., “Printer A. It is to be appreciated that
24 the request name may be in any format, provided it designates the Resource requested.

25 Upon receiving the request, each Peer Node evaluates the request (Block 208).
26 The process by which each Peer Node evaluates the request is explained with reference to
27 Figure 3 and is explained in greater detail hereinbelow. Each “up” Peer Node then
28 returns either an approval of the request or a denial (Block 210).

29 When the lock request is denied (Block 211), the lock request is subsequently
30 deleted from the Lock Broker Table for the First Node (Block 213) and the Client is
31 notified of the operation’s outcome (Block 217). Next, the process ends (Block 218).

1 The process will then restart when a subsequent lock request is received at a Node on the
2 Cluster.

3 When the lock request is approved, the lock request is designated as being in a
4 granted state and is entered into the First Node's Lock Broker Table as an active lock
5 (Block 212). The lock then remains as an active lock until a delete lock request is
6 received (Block 214). When the delete request is received, the lock is deleted from the
7 First Node's Lock Broker Table and the lock is removed from the Cluster (Block 216).
8 The Client is then notified of the operation's outcome (Block 217) and the process ends
9 (Block 218). Note, that for this embodiment no broadcast messages are needed to be sent
10 to any other Node when a lock is deleted from a Node's Lock Broker Table. Since each
11 Node stores only those locks requested by the CMLOCK daemon associated with the
12 Node, broadcast messages are not needed because data files at the Peer Nodes are not
13 updated when a lock is established or released. Peer Nodes do not maintain information
14 concerning locks on other Nodes and, instead, only maintain information relating to locks
15 held by that specific Node.

16 Referring now to Figure 4, a modification of the process flow of the embodiment
17 shown in Figure 2 is provided. In the embodiment shown in Figure 4, the present
18 invention supports pending lock requests, even if the Resource for which the lock is
19 initial requested has already been locked-up by another Node. As shown, this alternative
20 process flow closely resembles that of Figure 2 and utilizes the appropriate process steps
21 of Figure 3, as necessary. In order to illustrate the differences between the process flow
22 shown in Figure 2 and the process flow used in the embodiment illustrated in Figure 4,
23 those common steps are identified by the 200 series Block numbers utilized in Figure 2.
24 Additional process steps utilized in the second embodiment (and not utilized in the first
25 embodiment) are identified by 400 series Block numbers.

26 As shown, additional operations are provided in the second embodiment when a
27 lock is released (i.e., after Block 216 and before Block 217). These additional operations
28 provide that when a previously granted lock is released by a Node, the releasing Node
29 suitably notifies the Peer Nodes that the lock has been released (Block 420). This
30 notification is preferably accomplished utilizing a Cluster-wide broadcast message,
31 however, other Node-to-Node communications may also be utilized. Further, the
32 notification of the release of the previously granted lock is utilized in this embodiment to
33 signal other Nodes, which may or may not have a wait state request to lock-up a
34 particular Resource, that the Resource is now available for locking. However, it should

1 be noted that this process flow does not require the Node responsible for the Resource,
2 nor the Node which previously had the lock on the Resource to create a proxy of requests
3 for locking the Resource. Instead, each Node is responsible for maintaining wait stated
4 lock requests (i.e., a request to lock-up a Resource which has been denied, but for which
5 the Client is willing to wait for the Resource to become unlocked).

6 As shown in Figure 4, upon having a lock request denied, the process flow of this
7 embodiment supports “wait stated requests” by having the CMLOCK daemon associated
8 with the request issue at least one query to the Client as to whether the Client desires to
9 enter a waiting state, as necessary. However, for some requests, the CMLOCK daemon
10 may be pre-programmed to automatically enter a wait state, for example, when processing
11 a printing request (Block 422).

12 If the Client (or the CMLOCK daemon) does not desire to enter a wait state, the
13 process continues with deleting the lock request from the First Node (Block 213),
14 notifying the Client of the operation’s outcome (Block 217) and then the process ends
15 (Block 218).

16 If the Client (or the CMLOCK daemon) does desire to enter the wait state, the
17 process continues with the First Node awaiting notification, from the Peer Node currently
18 holding the lock, that the lock has been released and the Resource is now available for
19 “locking-up” (Blocks 424, 426) or notification from the Cluster Manager that a Cluster
20 membership change has occurred (Block 428). The Cluster membership may change
21 whenever a Node disconnects from the Cluster (for whatever reason) and any locks held
22 by that Node are then automatically released.

23 Further, the First Node may be configured to await notification that a lock has
24 been released, and a Resource becomes available, for only a specified period of time
25 (Block 430). If the Resource does not become available within the specified period of
time, the process then resumes with Block 213.

27 Additionally, it is to be appreciated, however, that the CMLOCK daemon utilizes
28 asynchronous operations. As such, the daemon is capable of processing other lock
29 requests by Clients, as well as, processing Peer Node requests and other tasks and
30 functions that may be performed by the daemon, as specified by the specific
31 implementation of the present invention. Such other processes include monitoring the
32 Cluster’s membership roster.

33 When the lock is released or the Peer Node’s Cluster membership changes, the
34 Node with the request in the waiting state receives the notification of the lock being

1 released or the Peer Node dropping off the Cluster. Then the process resumes with the
2 CMLOCK daemon at the First Node sending the lock request again to all the Peer Nodes
3 (Block 206). It is to be appreciated that in this embodiment, multiple Nodes may attempt
4 to simultaneously send requests for a Resource to all the other Nodes on the Cluster(s)
5 upon the release of a lock for a popular, or often desired, Resource. In the embodiment
6 shown in Figure 4, the first to send a lock request upon the release of a previously held
7 lock by another Node will be the first to lock up the Resource. Conflicts between
8 competing requests may be avoided by utilizing priority schemes based upon time of
9 wait, urgency of the request, Node hierarchy and other factors.

10 Additionally, it is conceivable that two Clients may desire to request the same
11 Resource with exclusive access at the same time. The present invention suitably
12 accommodates this situation by having every request received at a Node compared
13 against both the active and pending lock requests maintained in their respective Lock
14 Broker Tables. This process, for the above described scenario, would result in both lock
15 requests being denied by the other Client. Processing would then resume, for each Client,
16 with Block 211 (Figure 4). Then depending upon whether either or both Clients had
17 requested wait state processing, each request would be wait stated. Since neither Node
18 owned the lock, the process step in Block 426 would result in the NO path at which
19 instance the request would either await the changing of the other Node's Cluster
20 membership or the expiration of the timer, whichever occurred first.

21 However, in an alternative embodiment (not shown), the process of Figure 4 could
22 be modified such that a denial of a request due to a pending lock request of another Node
23 would be identified as such. The requesting Node could then be configured to
24 automatically resend the request based upon any of various factors. Such factors may
25 include waiting until the other Node dropped their respective pending request, utilizing a
26 priority scheme to determine which of the pending requests has priority, or based upon
27 any other process or schema.

28 As such, the embodiment provided in Figure 4 provides one embodiment of a
29 process for Nodes to request Resources and await their availability without requiring the
30 storage of lock requests at multiple Nodes. The present invention may be configured, as
31 desired, to utilize other process flows which depend upon a specification of a lock name,
32 an intent mode and a deny mode. Further, the approach shown in Figure 4 eliminates the
33 need for Clients to repeatedly send rejected requests by temporarily saving Requests (for

1 which a Client is willing to wait) in the Lock Broker Table as a pending or wait status
2 request.

3 Referring now to Figure 3, the process by which Nodes on a Cluster determine
4 whether a lock request is allowed or denied is shown. As shown, this process begins in
5 Figure 2 or Figure 4 at Block 208 with the receipt of the lock request at the Peer Node
6 (Block 300, Figure 3). Upon receiving the lock request, the Peer Node looks up whether
7 the lock name exists on their respective Lock Broker Table (Block 302). If the Lock
8 Broker Table does not contain the lock name, the Peer Node sends an approval back to
9 the First Node (i.e., the requesting Node) (Block 305) and the processing continues in
10 Figures 2 or 4 with Block 210.

11 If the Peer Node's Lock Broker Table does contain the lock name, the Peer Node
12 continues to examine the lock request. In this embodiment, the Peer Node continues to
13 examine the lock request by bit-wise exclusive OR'ing the integer value in the intent
14 mode for the lock request against the deny mode specified by the lock already existing in
15 the Peer Node's Lock Broker Table (i.e., the "intent mode/deny mode
16 comparison") (Block 306).

17 If the result of this bit-wise comparison is "NO" for any of the bits, the
18 comparison fails (Block 308). At which instance, the Peer Node sends a denial of the
19 request back to the First Node (i.e., the requesting Node) (Block 309). Processing then
20 continues in Figures 2 or 4 with Block 210 and following the "YES" path to Block 211.
21 Effectively, this "intent mode/deny mode comparison" determines whether the requested
22 use of the Resource requires a usage already denied by the existing lock. For example, a
23 Client might request to utilize a file in a read and write state. This request would conflict
24 with any other Client, already holding a lock, who also desires to read and write to the
25 file. Since, in this example, both Clients can not simultaneously have write authority, the
26 first to request the write is the lock holder and all subsequent requestors will be denied by
27 the lock holder.

28 Referring again to Figure 3, if the "intent mode/deny mode comparison" passes
29 (Block 308), the process continues with the Peer Node bit-wise comparing the deny mode
30 of the lock request against the intent mode of the existing lock (i.e., the "deny mode/intent
31 mode comparison") (Block 310). This comparison is also performed on a bit-wise basis
32 using an exclusive OR'ing methodology.

33 If the result of this bit-wise comparison is "NO" for any of the bits, the
34 comparison fails (Block 312). At which instance, the Peer Node sends a denial of the

1 request back to the First Node (i.e., the requesting Node)(Block 314). Processing then
2 continues in Figures 2 or 4 with Block 210 and following the “YES” path to Block 211.
3 Effectively, this “deny mode/intent mode comparison” determines whether the pending
4 lock request would deny the existing lock from utilizing the Resource in a manner already
5 intended by the existing lock. For example, a Client might request to utilize a display
6 monitor such that only their information is displayed on the screen. This request would
7 conflict with another Client, already holding an active lock, who also desires to display
8 information on the screen. Since the First Node’s lock request would disrupt the Peer
9 Node’s active lock, the Peer Node rejects the lock request.

10 Further, while the above described contention and evaluation processes utilized to
11 determine whether to approve a lock request are implemented using an exclusive OR’ing
12 bit-wise comparison of integer values, it is to be appreciated that the present invention is
13 not so limited. Other logical operations may be utilized that reach similar results.
14 Similarly, other comparisons may be utilized. For example, for certain Resources (such
15 as, a printer which may be either “in use” or “not in use”), the contention and evaluation
16 process may entail comparing only one status flag. Similarly, for more complex
17 operations (such as, data file sharing and manipulation operations), the contention and
18 evaluation process may be extremely complex and may involve factors other than
19 whether the Resource is currently being utilized. Such other factors may included, but are
20 not limited to: the anticipated amount of time that a specific Resource will be utilized for
21 a specific task; who is requesting the Resource (for example, certain Nodes may have
22 priority over other Nodes); and various other factors. In short, the present invention may
23 support any type of contention system that utilizes locally kept locks to determine
24 whether other Nodes on a Cluster may utilize a specific Resource in a given manner.

25 An example, of the beforementioned lock brokering techniques, is now provided
26 with reference to Figure 1. As shown in Figure 1, for Node A 102, two exemplary
27 pending lock requests are shown in the Lock Broker Table 126. The requests are for
28 “Printer A 104 and PC#1 116. As discussed previously herein, each of these requests are
29 suitably communicated to the Peer Nodes (i.e., Node B 108 and Node C 114) for the
30 Cluster 100.

31 Upon receiving the request, Node B examines its Lock Broker Table 128 and
32 determines whether a lock for the Resource exists. With respect to the request for Printer
33 A 104, neither Node B’s Lock Broker Table 128 nor Node C’s Lock Broker Table 130
34 contains a lock for the Printer A Resource. As such, this lock request is immediately

1 approved by both Node B and Node C (i.e., the answer to the query in Block 304 of
 2 Figure 3 is “NO”).
 3 With respect to the pending request for PC#1 116, this request is sent to each Peer
 4 Node. For Node B 108, its Lock Broker Table 128 does not contain a lock for the
 5 Resource identified as PC#1 116. As such, Node B returns an “Approved” status to the
 6 Node A lock request for PC#1 116. However, Node C’s Lock Broker Table 130 contains
 7 a lock for the same Resource. As such, the first query in Block 304 (Figure 3) returns a
 8 “YES” result such that the process continues with the “intent mode/deny mode
 9 comparison” previously described above.

10 Specifically, the Node C CMLOCK daemon compares, using an exclusive OR
 11 operation, the intent mode for the Node A PC#1 116 lock request and the deny mode for
 12 the Node C PC#1 116 active lock. Table 1, below, shows the result of this bit-wise
 13 comparison.

14 **Table 1**

PC#1 Intent Mode (i.e., pending lock request)	0	1	0	0
PC#1 Deny Mode (i.e., active lock)	1	0	0	1
RESULT (i.e., compares each column to ensure only a “1” is present in either row)	YES	YES	YES	YES

15
 16 Since the lock request passed the query by having each bit-wise comparison return
 17 a “YES” result (see Block 308, Figure 3), the process continues with the Node C 114
 18 CMLOCK daemon performing the “deny mode/intent mode comparison” (i.e., Block 310,
 19 Figure 3). Specifically, the Node C CMLOCK daemon compares, using an exclusive OR
 20 operation, the deny mode for the Node A PC#1 lock request and the intent mode for the
 21 Node C PC#1 active lock. Table 2, below, shows the result of this bit-wise comparison.

Table 2

Node A, PC#1 Deny Mode (i.e., pending lock request)	1	1	0	1
Node C, PC#1 Intent Mode (i.e., active lock)	1	0	1	0
RESULT (i.e., compares each column to ensure only a “1” is present in either row)	NO	YES	YES	YES

3 For this query, the lock request does not pass because the fourth bit of the lock
 4 request for Node A PC#1 deny mode and the fourth bit of the active lock request for the
 5 Node C PC#1 intent mode both are a “1”, which when exclusively “OR’ed” results in a
 6 “NO”. For this example, the NO indicates that a particular flag (designating a feature or
 7 aspect of the specified Resource) is desired to be exclusively set for the requesting lock
 8 while the active lock has already specified an intent to utilize the feature designated by
 9 the flag. As such, the Node A PC#1 pending lock request would be denied by Node C
 10 and processing would continue in Figure 3 with Block 309.

11 In another embodiment, the present invention also provides the capability of
 12 determining the status of the various locks held by any Node on a Cluster. In this
 13 embodiment, the Node desiring to know which locks are held by its Peer Nodes sends a
 14 request to each Peer Node. The request merely asks the Peer Node to identify those locks
 15 the Peer Node currently holds, including the lock name, the intent mode and the deny
 16 mode. Based upon this information, the requesting Node, via the CMLOCK daemon or
 17 another software routine, may create a “Held Locks” table or listing that identifies to the
 18 Client the locks held. Further, various other information may be stored in the Held
 19 Locks table, including, but not limited to, when a lock was initiated and when the lock is
 20 supposed to end. This information may be maintained by each Peer Node’s CMLOCK
 21 daemon or another software routine. This information may be supplied to the requesting
 22 Node’s Held Locks table upon request. As such, the present invention may be configured
 23 to provide an indication of the length of time all the various locks on the Cluster have
 24 been held and other information which may assist Cluster managers (whether human or
 25 automated) in determining “hang-ups” in the Cluster’s operations (for example, a “hung”
 26 print job that prevents other Clients on the Cluster from utilizing a printer). As such, the

1 present invention also facilitates the maintenance and trouble shooting of Clusters, Nodes,
2 and/or Resources via the lock request features provided by the CMLOCK daemon.
3 While the present invention has been described with reference to an illustrative
4 Cluster representation and various process flows, it is to be appreciated that larger or
5 smaller Clusters, more or less Nodes and or greater or fewer Resources may be utilized on
6 any system for which the present invention is utilized to establish and monitor locks on
7 Resources. Further, it is to be appreciated that fewer and/or greater steps may be utilized,
8 as desired, in the process flows of the present invention. As such, the present invention is
9 to be construed as covering those subject matters discussed supported herein in addition
10 to those equivalents thereof as expressed in the following claims.